UNCLASSIFIED

AFIT/EN/TR95-02

Air Force Institute of Technology

On a Distributed Anytime Architecture for Probabilistic Reasoning

Eugene Santos Jr. Solomon Eyal Shimony Edward Williams

April 21, 1995

Approved for public release; distribution unlimited

On a Distributed Anytime Architecture for Probabilistic Reasoning

Eugene Santos Jr.† Solomon Eyal Shimony‡
Edward Williams†

April 21, 1995

†Dept. of Elec. and Computer Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH 45433-7765, esantos@afit.af.mil and ewilliam@afit.af.mil

‡Dept. of Math. and Computer Science, Ben Gurion University of the Negev, 84105 Beer-Sheva, Israel, shimony@cs.bgu.ac.il

Abstract

An architecture for unifying various algorithms for probabilistic reasoning is presented. Any algorithms having anytime, anywhere characteristics may be mixed in this scheme. Since algorithms for probabilistic reasoning have widely different behavior over classes of Bayes networks, the scheme permits taking advantage of the set of algorithms that happen to perform well for the problem instance at hand. We concentrate on belief updating and belief revision. Some results are presented for our system (OVERMIND) consisting of several genetic algorithm instances, A*, etc. running in parallel.

Keywords: Probabilistic Reasoning, Anytime Algorithms, Anywhere Algorithms, Parallelized Algorithms, Meta-Reasoning, Deliberation Scheduling.

1 Introduction

To satisfy the ever increasing demand for fast inferencing, especially in highly dynamic tasks such as real-time planning and scheduling [12, 11], the ability to provide a near optimal solution at any given moment is extremely important. The capability of improving upon the solutions as more time and resources are allocated is a natural way in which to continually revise and update our

[1]This research was supported in part by AFOSR Project #940006.

operations/conclusions. Algorithms which have this property of producing a solution at any point in time are called "anytime" algorithms [3, 11, 18].

Another factor that has a major impact on reasoning is that of uncertainty. It is very rare to find something which is completely true or false. There always seem to be exceptions to whatever rule we can come up with. Trying to encode all possible situations as "if-then" rules, we often find our resulting knowledge base to be unacceptably large and impossible to work with. At other times, it may just be the case that information is currently missing or unavailable. Unfortunately, adding uncertainty to the reasoning process results in adding significant complexity as well [9, 32, 8].

Parallelism and distributed processing are ways of alleviating the time-complexity of a problem at the cost of additional hardware. However, in order to exploit parallelism, the tasks in the distributed environment must be able to exploit intermediate results produced by the other components of the system. Algorithms with this property are called "anywhere" algorithms. When different algorithms having both the anytime and anywhere properties are harnessed together into a cooperative system, the resultant architecture can exploit the best characteristics of each algorithm.

In this paper we present such an architecture for a fast inference engine. This architecture, called OVERMIND can be readily implemented using existing packages such as PVM [14] and GENESIS [16]. The reasoning engine discussed here, as part of a larger system called PESKI [27] is currently being used for engine diagnostics for the Space Shuttle Program [2]. However, for the purposes of this paper, we concentrate mainly on the reasoning architecture. The rest of this paper provides some background on probabilistic reasoning, presents the OVERMIND architecture, describes our current implementation and results, and concludes with plans for future work.

2 Background: Probabilistic Reasoning

For the sake of simplicity, we focus here exclusively on Bayesian networks [21]. Other models have been shown to be very closely related in computational difficulty and style to Bayesian networks [7, 31, 6, 26].

In probabilistic reasoning, random variables (abbreviated, r.v.) are used to represent events and/or objects in the world. By making various instantiations to these r.v.s, we can model the current state of the world. This will involve computing joint probabilities of the given r.v.s. Unfortunately, the task is nearly impossible without additional information concerning relationships between the r.v.s. In the worst case, we would need the probabilities of every instantiation combination which is combinatorially explosive to say the least.

On the other hand, we look at the chain rule as follows:

$$P(A_1, A_2, A_3, A_4, A_5) = \tag{1}$$

$$P(A_1|A_2, A_3, A_4, A_5)P(A_2|A_3, A_4, A_5)P(A_3|A_4, A_5)P(A_4|A_5)P(A_5).$$

Bayesian networks [21] take this a step further by making the important observation that certain r.v. pairs may become uncorrelated once information concerning some other r.v.(s) is known.[2] More precisely, we may have the following independence condition:

$$P(A|C_1, \ldots, C_n, U) = P(A|C_1, \ldots, C_n) \tag{2}$$

for some collection of r.v.s U. Intuitively, we can interpret this as saying that A is determined by $C_1, \ldots, C_n$ regardless of U.

Combined with the chain rule, these conditional independencies allow us to replace the terms in (1) with the smaller conditionals (2). Thus, instead of explicitly keeping the joint probabilities, all we need are smaller conditional probability tables which we can then use to compute the joint probabilities. What we have is a directed acyclic graph of r.v. relationships. Directed arcs between r.v.s represent conditional dependencies. When all the parents of a given r.v. are instantiated, that r.v. is said to be conditionally independent of any ancestors given its parents (equation (2)).

There are two types of computations performed with Bayesian networks: *belief updating* and *belief revision* [21]. Belief revision is best used for modeling explanatory/diagnostic tasks. Basically, some evidence or observation is given to us, and our task is to come up with a set of hypothesis that together constitute the most satisfactory explanation/interpretation of the evidence at hand. This process has also been considered *abductive reasoning* in one form or another [28, 7]. More formally, if W is the set of all r.v.s in our given Bayesian network and e is our given evidence[3], any complete instantiations to all the r.v.s in W which is consistent with e will be called an *explanation* or *interpretation* of e. Our problem is to find an explanation w^* such that

$$P(w^*|e) = \max_w P(w|e). \tag{3}$$

Intuitively, we can think of the non-evidence r.v.s in W as possible hypothesis for e.

Belief updating, on the other hand, is mainly interested in determining the marginal probabilities of a particular r.v. given some evidence. We can interpret this as updating our beliefs about the r.v. when given new observations about the world. Different from belief revision, our problem is to compute $P(A = a|e)$ for all possible instantiations a where A is our r.v. of interest. We note, though, that belief updating can also be computed using belief revision [19, 30].

As an example, consider the small Bayesian network in Figure 1; to the right of the network are the associated probabilities. If we are given the observation that node D is true, it is our task under belief revision to determine the

[2] We assume that each r.v. has a finite number of instantiations.

[3] That is, e represents a set of instantiations made on a subset of W.

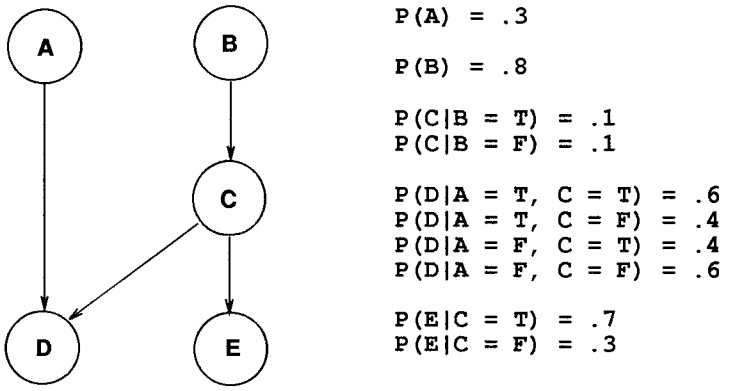

Figure 1: A small example Bayesian network

most probable complete instantiation over all the nodes in the network which is consistent with D being true and maximizes (3). This results in the solution $\{A = F, B = T, C = F, D = T, E = F\}$ which has joint probability 0.21168. For belief updating, if B is our r.v. of interest, we would compute the following: $P(B = T|D = T) = 0.4256$ and $P(B = F|D = T) = 0.5744$.

With a small network such as this, a valid solution method is to simply tabulate all the possible values of the r.v.s and then calculate the probabilities manually. Once the network gets larger and more complex, this method is obviously unacceptable and more efficient methods must be employed.

Current methods for *approximating* belief revision and updating on Bayes nets are best-first search, linear programming and genetic algorithms. (*Exact* algorithms for these problems abound which are too numerous to cite here, see [21]. However, these are, as a rule, exponential-time algorithms.) The first two methods are deterministic: they will find the optimal solution, given enough time and resources. Solutions based on genetic algorithms are nondeterministic.

It was shown in [7] that belief revision can be solved using a best-first or A^* search strategy. Hence, the problem of network topology is absorbed in the choice of a search heuristic. Furthermore, through back-tracking, the alternative solutions could be generated. Another class of techniques based on linear programming has been shown to compute the most-probable explanation more efficiently than other approaches [26]. By transforming the problem into 0-1 integer linear programming (ILP), highly efficient tools and techniques such as the Simplex method and Karmarkar's projective scaling algorithm can be applied.

Nondeterministic approaches such as genetic algorithms [23] are being studied since they seem to promise near-optimal solutions in relatively short time. However, there is no guarantee of finding the most probable solution or even

achieving some sort of reasonable error bound. Recently, [29] demonstrated that the techniques described above for belief revision can also be adapted to efficiently perform belief updating.

While the general problem of reasoning on Bayes nets is NP-hard [9, 32, 8], each of the above algorithms exploit *different characteristics* of the problem domain to reduce the time required to generate the solution. That is, each is potentially exponential, but handles certain *different kinds of networks* efficiently. The OVERMIND architecture presented in the next section pulls together these different methods and attempts to exploit the best qualities of each to efficiently perform the required task.

3 The OVERMIND architecture

Our architecture for a reasoning engine consists of three major components:

- **Intelligent Resource Allocator** (IRA) - manages and allocates available computing resources such as multi-processors, workstations, and personal computers.
- **Overseer Task Manager** (OVERSEER) - directs the flow of messages/information between co-operative tasks and initiates new tasks as needed.
- **Library of Tasks** (LOTS) - contains a wide variety of tasks/algorithms suitable for performing various reasoning computations.

As can be seen from Figure 2, the OVERMIND architecture is geared towards anytime solutions, anywhere solution sharing and co-operative task management in order to provide fast near optimal solutions as well as use networks of computing resources effectively. Much of the system is constructed from existing packages and current computing resources in the form of networked workstations and personal computers as shown in the next section.

3.1 Library of Tasks

First, we consider the library of tasks, LOTS, in more detail. Clearly, from our discussions in Section 2, reasoning with Bayesian networks is quite difficult. Although there are a number of approaches available, only a few of them address the issue of anytime computations [10, 33] and none have been combined together to work in a cooperative fashion. Exact algorithms are generally exponential-time, but are efficient for certain classes of network (usually dependent on topology). Nondeterministic methods, on the other hand, are usually anytime, but cannot guarantee that they will reach the correct solution in finite time. Their quality of approximation usually depends on the distributions, but is mostly independent of topology. This suggests that the best algorithm to use is problem-instance dependent.

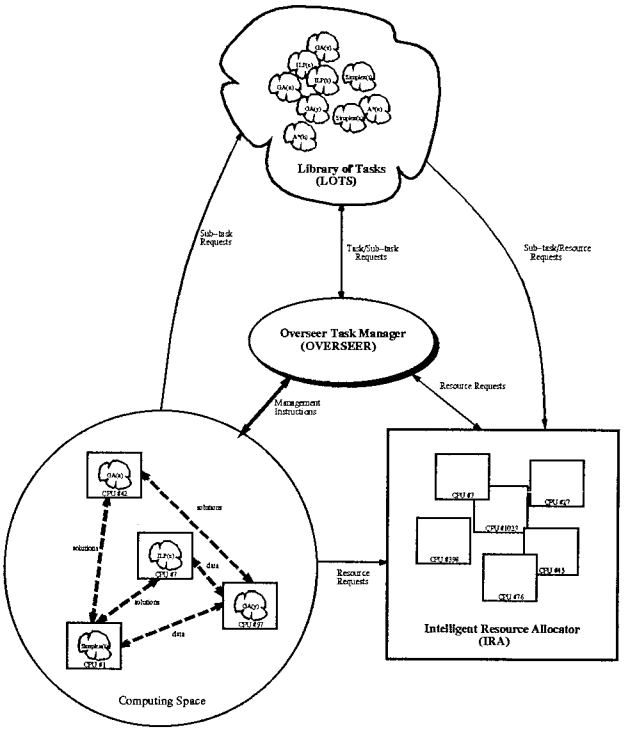

Figure 2: The OVERMIND architecture.

In terms of problem suitability, it therefore seems natural to choose the best approach per problem instance dynamically. Better yet, if a particular approach is good at starting off a problem or solving some portion of it we can then take its partial solution and pass it to another approach which itself may be better on this new portion to work on. This naturally leads us in a direction amenable to anytime and anywhere solutions. LOTS needs to contain algorithms exhibiting both of these important properties, so that solution sharing between methods is possible.

Up to this point, we have been referring to the individual algorithms as if they were single processes on a workstation. There is nothing in this architecture to prevent the major tasks from actually consisting of a group of parallel processes executing on multiprocessor systems. In fact, existing systems for genetic algorithms such as GENESIS [16], integer linear programming [4, 1] and A^* [17] searches can be naturally decomposed and distributed.

3.1.1 Genetic Algorithms

As we mentioned earlier, the class of solutions based on genetic algorithms [15, 23] is a nondeterministic approach to reasoning with Bayesian networks. Genetic algorithms take a small sample from the space of possible solutions and uses it to generate other (possibly better) solutions. The method of generating new solutions is modeled after natural genetic evolution. Each pass through a sequence of operations on the current solution set is called a generation. Over time, the hope is that the best solutions survive through to the next generation and contribute pieces to form even better solutions. While genetic algorithms effectively blanket the search space through swapping pieces between elements, they have problems in determining whether the best solution so far is actually the optimal solution. There is no stopping criterion that guarantees an optimal answer. However, its ability to generate solutions early can serve as a starting point if possible for other deterministic methods.

The process of evaluating the members of the population inherently produces a number of intermediate solutions – the best of these solutions is easily saved and distributed, exactly what we need for an anytime algorithm. The solutions received from other tasks are easily utilized by simply inserting them into the population between generations, so genetic algorithms also have the anywhere property.

3.1.2 Linear Programming

At the core of this approach is transforming the search for the most probable explanation in belief revision into an *integer linear programming* problem [20]. The structure found in a Bayesian network B and the evidence $\mathcal{E}$ are mapped into a corresponding linear optimization problem $L(B, \mathcal{E})$ (consisting of the system of linear constraints and the objective function defined in [26]); such that

solutions satisfying the constraints will correspond to valid explanations by a mapping function a (see [26]), and vice versa. Hence, properties of explanations, such as each r.v. is instantiated to exactly one value, is preserved within the linear system. The optimal integral solution of the system uniquely determines the most probable instantiation (paraphrased from [26]):

Theorem 1 *Let s be a integral solution to $L(B, \mathcal{E})$. Then $a(s)$ is an assignment to B that is consistent with the evidence. Furthermore, if s is an* optimal *integral solution to $L(B, \mathcal{E})$, then $a(s)$ is a most probable instantiation of B given the evidence.*

Once the transformation is completed, we can then solve the problem using powerful and efficient algorithms for linear programming such as the Simplex method and Karmarkar's projective scaling algorithm [20]. This approach proved to be fairly efficient at performing belief revision [26].

The biggest problem in computing the optimal solution lies in the requirement that the numerical solutions generated be integral due to the discrete nature of Bayesian networks (hence, integer linear programming as opposed to linear programming). However, the key observation in [26] is as follows: For each integral solution satisfying our linear constraints, there are a great number of "better" non-integral solutions. As long as there is a better solution (integral or not), this forces us to continue searching for a better solution even though we might currently be looking at the best integral one. This is the crux of linear programming when we are looking for integer solutions.

Clearly, the *number* of integral solutions is quite large. In fact, it is combinatorial in the number of possible r.v. instantiation. Fortunately, work done in graph theory indicates that integer programming problems derived from graph problems tend to move the Simplex process towards integral solutions [13] when optimality (probability) is not considered.

Instead of searching directly for the optimal integral solution, we eliminate the optimality condition and simply search for some integral solution which satisfies our constraints. Obviously, we have no guarantees that this solution will be optimal, but the process is fairly efficient. Combining this with the introduction of linear constraints which bound the space of solutions to those with certain probability values, we now search for an integral solution with a specific upper and lower bound on solution quality. Then by sliding these bounds, in this case upwards, we will move towards finding the optimal solution. The optimal can be identified when no integral solutions exist which have a better probability.

As we can see, [26] can be modified to generate intermediate solutions in anytime fashion, which are then used as seeds for the generations of the genetic algorithm. Furthermore, the genetic algorithm inherently produces intermediate solutions that are usable by the linear programming algorithm to provide viable "jump-start" solutions as well as to prune its solution space more rapidly [25].

3.1.3 Best-First Search Techniques (A*, etc.)

Traditional best-first algorithms, such as A^*, begin with an initial state (usually an instantiation containing only the evidence, for this problem), and perform heuristic (possibly exhaustive) search for the optimal goal (here: complete instantiation). Two heavily problem-dependent items are the next-state generator (here: assign several more variables, in various ways), and the heuristic evaluation function. Earlier work experimented with different admissible heuristics for this problem: current cost [7], and shared cost [5]. Both are guaranteed to provide the correct optimum, given unlimited resources (modified from [7, 5]):

Theorem 2 *The best-first search algorithm, with either the cost-so-far or shared-cost heuristic, will always halt with the highest probability instantiation consistent with the evidence. If resumed at the halting point, the algorithm will enumerate the instantiations in decreasing order of probability.*

In the context of this paper, we need to be able to interface this algorithm with other algorithms, and provide anytime-anywhere characteristics. We begin by noting that in limiting resources, there is no guarantee that the algorithm will give us the optimal answer. In this case, therefore, there is added incentive in using heuristics that give better pruning at the cost of admissibility. However, most important is the need for the following capabilities:

1. Provide an approximate answer(s) when interrupted. (*anytime*)
2. Allow the algorithm to accept initial guesses from another source. (*anywhere*)

Of these requirements, the first is by far the easier. To provide an answer, simply pick the currently best partial instantiation, and if necessary, complete the instantiation (for example, by random instantiation of the unassigned variables). The completion can also be achieved by selecting local optima for variable instantiations, for example by using the efficient polytree belief revision algorithm [21] or by any other fast scheme. If best-first is used to provide populations for GA, several partial solutions from the agenda of the search algorithm can be used.

Allowing an initial guess to start off best-first is more difficult. The initial guesses have to be compatible with partial instantiations, preferably such that the optimal solution be reachable from these guesses. Here, we are not assured of this property. Rather, we use best-first to find the most probable complete instantiation among those compatible with the guess. To be useful, the guess should not be a complete instantiation, so need to assume that other components in the system have search states that are partial instantiations, or provide a mechanism for making partial instantiations from the complete instantiations. Genetic algorithms and ILP usually operate on complete assignments, and thus

in their intermediate solutions we need to "un-assign" several variables before the solutions are passed to A^*. We intend to do that by randomized methods, and this fits in well with GA. Alternately, it is possible to modify both the GA and the ILP to work on partial assignments [29].

3.2 Intelligent Resource Allocator

Next, our Intelligent Resource Allocator, IRA, will serve to maximize processor use by coordinating requests for resources from OVERSEER and even possibly the tasks themselves. Currently, we utilize a network of workstations and allocate single processors mainly to the major tasks such as genetic algorithms, integer linear programming and A^*. We also recognize that there are differences in workstation/computer configurations (memory, CPU speed, I/O bandwidth, etc.) and that the different tasks each have specific resource requirements. For example, the GAs are single CPU intensive but require little memory while certain ILP tasks require multi-processing. Simply identifying a Sparc2 from a Sparc20 can improve efficiency.

For the most part, we can predict relatively easily the resource requirements for our current collection of tasks. However, once we begin introducing highly specialized tasks and sophisticated variants of our existing tasks, we must move towards a model-theoretic approach of resources and requirements management.

3.3 The Overseer

In conjunction with IRA, the Overseer Task Manager (OVERSEER) handles the message trafficking between tasks. Currently, OVERSEER plays a simple messenger role enforcing synchronized activity between the tasks. Even in this simple scheme, the loosely coupled architecture allows us to utilize multiple instances of the same major task with different operating parameters. Packages such as the Parallel Virtual Machine (PVM) [14] are used to manage communications and synchronization between tasks as well as allocate resources. They easily form the starting foundations for both IRA and OVERSEER.

Advanced capabilities will involve determining which tasks are promising, so that more time and hardware can be allocated to them, as outlined below. Employing meta-reasoning to consider what computational tasks to execute is known as *deliberation scheduling* [12, 11]. The overseer will essentially implement deliberation scheduling on multiple computational resources. A decision-theoretic framework is used, in which a utility function over outcomes is given. To operate within this framework, some form of distribution over outcomes is needed.

In real-time probabilistic reasoning, a reasonable goal would be to get a good approximation in minimum time, while ensuring *some* reasonable approximation before a deadline elapses. Utility would thus be a decreasing function of time and error. To enable computation of expected utility of a candidate deliberation

schedule, some estimate of runtime and quality of results should be available for each algorithm. We next discuss how to obtain these estimates for best-first search and ILP techniques.

3.3.1 Expected Utility Estimates for Belief Updating

We assume that we are computing the posterior probability (given evidence e) of a single query node q, with domain values q_i for i between 1 and d, the number of possible states for node q. The best-first algorithm searches for terms or assignments, and collects their probability mass to estimate the marginal probability, as in [29, 22]. Assume that there is one agenda for each state of q, and one agenda for the negation of the evidence. Let $\hat{P}(q_i e)$ be the current mass collected for the joint probability of q_i and e. Let ϵ be the remaining mass, i.e. mass in assignments not yet enumerated by the algorithm. $\hat{P}(e)$ is the sum of all the $\hat{P}(q_i e)$.

Following [29, 22], the bounds on the conditional probabilities are:

$$\frac{\hat{P}(q_i e) + \epsilon}{\hat{P}(e) + \epsilon} \geq P(q_i|e) \geq \frac{\hat{P}(q_i e)}{\hat{P}(e) + \epsilon} \tag{4}$$

Suppose that we consider getting the next instantiation from some agenda, and suppose that the (currently unknown) instantiation has effective (i.e. non-overlapping with previous instantiations) probability δ. We can now update the probability bounds as follows. There are three cases: (1) instantiation is inconsistent with the evidence, (2) instantiation is consistent with evidence and q_i, (3) instantiation is consistent with some *other* state q_j (and thus inconsistent with q_i). The resulting updated bounds are, respectively:

$$\frac{\hat{P}(q_i e) + \epsilon - \delta}{\hat{P}(e) + \epsilon - \delta} \geq P(q_i|e) \geq \frac{\hat{P}(q_i e)}{\hat{P}(e) + \epsilon - \delta} \tag{5}$$

$$\frac{\hat{P}(q_i e) + \epsilon}{\hat{P}(e) + \epsilon} \geq P(q_i|e) \geq \frac{\hat{P}(q_i e) + \delta}{\hat{P}(e) + \epsilon} \tag{6}$$

$$\frac{\hat{P}(q_i e) + \epsilon - \delta}{\hat{P}(e) + \epsilon} \geq P(q_i|e) \geq \frac{\hat{P}(q_i e)}{\hat{P}(e) + \epsilon} \tag{7}$$

From these bounds, we can compute the new RMS (root mean square) error for all values of i, for each possible selected agenda: For case (1) it reduces (after some algebraic tweaking) to:

$$e^1_{RMS} = \frac{\sqrt{d}}{\hat{P}(e) + \epsilon}(\epsilon - \delta) \tag{8}$$

while for instantiations consistent with the evidence we get:

$$e^{2,3}_{RMS} = \frac{\sqrt{d}}{\hat{P}(e) + \epsilon - \delta}(\epsilon - \delta) \tag{9}$$

It is reasonable, under the meta-greedy and single-step assumptions [24], to make the utility of the search step a function of the reduction in RMS error, which is:

$$\Delta e^{1}_{RMS} = e^{old}_{RMS} - e^{1}_{RMS} = \frac{\sqrt{d}}{\hat{P}(e) + \epsilon}\delta \tag{10}$$

$$\Delta e^{2,3}_{RMS} = e^{old}_{RMS} - e^{2,3}_{RMS} = \frac{\sqrt{d}}{(\hat{P}(e) + \epsilon)} \frac{\delta \hat{P}(e)}{(\hat{P}(e) + \epsilon - \delta)} \tag{11}$$

The latter equation cannot be further simplified, since $\delta < \epsilon$, but none of these quantities are negligible w.r.t. $\hat{P}(e)$ or each other, in general. The remaining problem here is twofold: (1) we do not know δ before the search step is applied, and (2) in our system, time has an impact on utility, and we do not know how long the search step will take. Nevertheless, these can be estimated: we estimate δ by the heuristic value of the best agenda item, and the search step time by the execution time of previous search steps. The "expected" utility is the decrease in RMS error divided by expected execution time, or any other such function that takes into account the value of information over time.

Since the discussion of error estimates for best-first search is unspecific w.r.t. the actual search algorithm, the equations apply to any algorithm that approximates the probabilities by collecting mass of terms and instantiations. The equations are easily adapted to the case where the search uses only one agenda, by approximating the probability that the next instantiation will be of a particular class of the above three. In particular, they apply to the ILP version of the algorithm.

4 Implementation and Results

PVM [14] was used as the IRA to manage the distribution of tasks across hardware platforms. The genetic algorithms were either existing applications that were adapted to this environment, or were developed using GENESIS [16] as the foundation. Finally, the A* and ILP were developed from existing code in [26, 30] but could have easily used packages such as MINOS.

With PVM providing the framework, only a minimal amount of C++ coding was needed to get the system running. The OVERSEER was also created to spawn the tasks through PVM, pass them their operating parameters and datasets, and then relay intermediate solutions as they were generated.

The first genetic algorithm was implemented on top of GENESIS, a generic framework for implementing genetic algorithms. First, the framework was

adapted to work under C++, then the additional functions necessary to manipulate Bayesian Networks were added. The framework was also modified to allow it to run under PVM and to accept its operating parameters from the IRA instead of a control file. Similarly, the ILP and A* tasks were modified by introducing various hooks for PVM.

The platform for this implementation was a network of Sun Workstations consisting of Sparc2s and Sparc20s. It would take only a minimum amount of effort to port this implementation to any architecture supported by PVM, including systems from SGI, DEC, and Intel x86-based PCs running some form of Unix. The OVERMIND architecture could also be effective running on a single workstation, if necessary.

For our experiments, we measured solution quality versus time. Initial testing was performed using multiple instances of GAs. As we suspected, two GAs, especially with different parameters, arrived at the same solution quality faster, typically 20% faster. Additional GA tasks increased the speed even further.

The other algorithms were then added to a base 2 GA system. First was HySS, a hybrid stochastic simulation algorithm not discussed in this paper, which resulted in an $3-5\%$ speed increase, then the A* search and ILP for an additional $15-25\%$ increase.

When the other algorithms were added, not only did we reach our solutions faster, but we observed the strengths of the different methods as they combined together. The A* tended to produce reasonable solutions immediately, GAs took those solutions near some maximas, HySS fine-tuned those maximas, and the ILP finished the optimization and generated the optimal solution.

5 Conclusions

The OVERMIND architecture is a first step towards providing a practical probabilistic reasoning system. It exploits the anytime, anywhere properties of disparate reasoning algorithms such as GAs, ILP and A* and unifies them into a single model of computation which is naturally parallelizable. By insuring co-operation between these algorithms through approaches such as deliberation scheduling, OVERMIND can exploit the best characteristics of each algorithm.

OVERMIND can be built on our typical computing hardware such as networks of workstations, etc. and requires minimal modification of existing software packages. The biggest amount of work lies in collecting the various algorithms/tasks for the system to manage. We note though that OVERMIND is actually general enough to work in any domain provided such a library of tasks is available as with probabilistic reasoning.

Our experimental results seem very promising and have served to support some of our basic expectations for the architecture. Among the future work we foresee is an even tighter co-operation between the reasoning tasks and the impact of increased communications overhead, the parameterization of tasks

and its variants such as GAs and how to dynamically generate a "promising" task, and to continue the study of various approaches for task management and resource allocation.

References

[1] Paul D. Bailor and Walter D. Seward. A distributed computer algorithm for solving integer linear programming problems. In *Proceedings of the Fourth Conference on Hypercubes, Concurr ent Computers and Applications*, pages 1083–1088, 1989.

[2] Darwyn Banks. Acquiring diagnostic knowledge of the ssme hpotp for bayesian forests. Master's thesis, Graduate School of Engineering, Air Force Institute of Technology, 1995 (in progress).

[3] Mark Boddy. Anytime problem solving using dynamic programming. In *Proceedings of the AAAI Conference*, pages 738–743, 1991.

[4] Rochelle L. Boehning, Ralph M. Butler, and Billy E. Gillett. A parallel integer linear programming algorithm. *European Journal of Operational Research*, 34:393–398, 1988.

[5] Eugene Charniak and Saadia Husain. A new admissible heuristic for minimal-cost proofs. In *Proceedings of the AAAI Conference*, pages 446–451, 1991.

[6] Eugene Charniak and Eugene Santos, Jr. Dynamic map calculations for abduction. In *Proceedings of the AAAI Conference*, pages 552–557, 1992.

[7] Eugene Charniak and Solomon E. Shimony. Cost-based abduction and MAP explanation. *Artificial Intelligence*, 66:345–374, 1994.

[8] Gregory F. Cooper. Probabilistic inference using belief networks is NP-hard. Technical Report KSL-87-27, Medical Computer Science Group, Stanford University, 1987.

[9] Paul Dagum and Michael Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60 (1):141–153, 1993.

[10] Bruce D'Ambrosio. Incremental probabilistic inference. In *Uncertainty in AI, Proceedings of the 9th Conference*, July 1993.

[11] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings of the AAAI Conference*, pages 49–54, 1988.

[12] Thomas Dean and Michael Wellman. *Planning and Control.* Morgan Kaufmann, 1991.

[13] Robert S. Garfinkel and George L. Nemhauser. *Integer Programming.* John Wiley & Sons, Inc., 1972.

[14] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Mancheck, and Vaidy Sunderam. PVM 3 User's Guide and Reference Manual. Technical Report ORNL/TM-12187, Oak Ridge National Laboratory, 1994.

[15] David E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[16] John J. Grefenstette. A User's Guide to GENESIS. Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory, Washington, DC, 1987.

[17] Michael S. Gudaitis, Gary B. Lamont, and Andrew J. Terzuoli. Multicriteria vehicle route-planning using parallel A* search. In *Proceedings of the ACM Symposium on Applied Computing,* 1995.

[18] Eric J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the Workshop on Uncertainty in Artificial Intelligence,* 1987.

[19] Jak Kirman, Ann Nicholson, Moises Lejter, Eugene Santos, Jr., and Thomas Dean. Using goals to find plans with high expected utility. In *Proceedings of the Second European Workshop on Planning,* 1993.

[20] George L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors. *Optimization: Handbooks in Operations Research and Management Science Volume 1,* volume 1. North Holland, 1989.

[21] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, CA, 1988.

[22] David Poole. The use of conflicts in searching Bayesian networks. In *Uncertainty in AI, Proceedings of the 9th Conference,* July 1993.

[23] Carlos Rojas-Guzman and Mark A. Kramer. GALGO: A Genetic ALGOrithm decision support tool for complex uncertain systems modeled with bayesian belief networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence,* pages 368–375, 1993.

[24] Stuart Russel and Eric Wefald. On optimal game-tree search using rational meta-reasoning. In *Proceedings 11th IJCAI,* pages 334–340, August 1989.

[25] Eugene Santos, Jr. Efficient jumpstarting of hill-climbing search for the most probable explanation. In *Proceedings of International Congress on Computer Systems and Applied Mathematics Workshop on Constraint Processing*, pages 183–194, 1993.

[26] Eugene Santos, Jr. A fast hill-climbing approach without an energy function for finding mpe. In *Proceedings of the 5th IEEE International Conference on Tools with Artificial Intelligence*, 1993.

[27] Eugene Santos, Jr. A fully integrated probabilistic framework for expert systems development. AFOSR Grant #940006, 1994.

[28] Eugene Santos, Jr. A linear constraint satisfaction approach to cost-based abduction. *Artificial Intelligence*, 65(1):1–28, 1994.

[29] Eugene Santos, Jr. and Solomon E. Shimony. Exploiting case-based independence for approximating marginal probabilities. *Submitted to IJAR*, 1994.

[30] Eugene Santos, Jr. and Solomon Eyal Shimony. Belief updating by enumerating high-probability independence-based assignments. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 506–513, 1994.

[31] Solomon E. Shimony. The role of relevance in explanation I: Irrelevance as statistical independence. *International Journal of Approximate Reasoning*, June 1993.

[32] Solomon E. Shimony. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, 68:399–410, 1994.

[33] Michael P. Wellman and Chao-Lin Liu. State-space abstraction for anytime evaluation of probabilistic networks. In *Uncertainty in AI, Proceedings of the Tenth Conference*, pages 567–574, July 1994.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1995	Technical Report

4. TITLE AND SUBTITLE
ON A DISTRIBUTED ANYTIME ARCHITECTURE FOR PROBABILISTIC REASONING

5. FUNDING NUMBERS

6. AUTHOR(S)
Eugene Santos Jr., Solomon Eyal Shimony, and Edward Williams

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/EN/TR95-02

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

An architecture for unifying various algorithms for probabilistic reasoning is presented. Any algorithms having anytime, anywhere characteristics may be mixed in this scheme. Since algorithms for probabilistic reasoning have widely different behavior over classes of Bayes networks, the scheme permits taking advantage of the set of algorithms that happen to perform well for the problem instance at hand. We concentrate on belief updating and belief revision. Some results are presented for our system (OVERMIND) consisting of several genetic algorithm instances, A*, etc. running in parallel.

14. SUBJECT TERMS
Probabilistic Reasoning, Anytime Algorithms, Anywhere Algorithms, Parallelized Algorithms, Meta-Reasoning, Deliberation Scheduling

15. NUMBER OF PAGES
18

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL